Programming 101

This resource will cover the basics of programming in *FIRST*® Robotics Competition. It covers C++, Java/Kotlin and LabVIEW.

Level One: Getting Your Robot Running

1. ***Picking a Programming Language: Java, C++, or LabVIEW***
   - **Java** is a textual language that is commonly taught at high schools and used for the AP CS exams. It is a "safe" language in that runs in its own virtual environment. While it doesn't generally affect *FIRST* Robotics Competition, this virtual environment, also known as the JVM, means that Java programs are noticeably slower than compiled languages when used for computationally intensive tasks. Java is often selected because of its ease of use, and cross-compatibility. Teams that use Java include 254, 125, 503, 4911, and 1241.
   - **C++** is a fast textual programming language. It is used in industry for real time systems because of its power and efficiency, but the learning curve is much steeper than Java. C++ evolved from the C programming language and the mixture of historical and modern features sometimes lead to confusing syntax and/or unexpected behaviors. *FIRST* Robotics Competition teams primarily use it due to its speed, flexibility, and its extensive mathematical libraries. Teams that use C++ include 971 and 1678.
   - **LabVIEW** is a graphical dataflow programming language developed by National Instruments (NI) for use by engineers and technicians. The LEGO WeDo and Mindstorms languages used for *FIRST* LEGO League Jr and *FIRST* LEGO League are derivatives of LabVIEW; so students coming from those programs may find it familiar. In a LabVIEW diagram, it is very easy to take advantage of advanced computing features, such as running pieces of code in parallel. While powerful, such features often introduce new issues to deal with. NI provides extensive debugging tools, however. The LabVIEW environment and language come with its own learning curve and unique challenges. *FIRST* Robotics Competition teams primarily use it due to its simplified graphical syntax and extensive engineering libraries. Teams that use LabVIEW include 33, 359, 624, 1986, and 2468.
   - **Choosing what language always depends on what is easiest for *your* team.** For example, it can often make sense to choose a language because a programming mentor is an expert in that language. On the other hand, it might make sense to choose based on the ease of learning a given language. No matter what your decision, remember that the choice of programming language is specific to the working environment and people of your team. All of the

languages are capable, well supported, and sufficiently powerful for *FIRST* Robotics Competition use.

2. ***Teaching the Programming Language***
   - When teaching programming for *FIRST* Robotics Competition, there are two distinct subjects that need to be taught. The first is the semantics and syntax of the programming language itself, and the second is interfacing with *FIRST* Robotics Competition components. A guide on learning the C++ language can be found here, Java here, and LabVIEW here.

3. ***Picking your starting code***
   - For Java and C++, there are four different “classes” that can be used when interfacing with the Robot. A comparison of these classes, and what they mean can be found here.
   - For LabVIEW, the starting templates include a mixture of timed, iterative, and spawn/abort mechanisms. The templates are described here.

4. ***Once your team has picked a programming language and has started coding, the FIRST Robotics Competition Docs site is a good resource on how to set up your development environment and get code onto your robot. These guides are invaluable for FIRST Robotics Competition programming.***
   - Getting Started
   - C++\Java
   - LabVIEW

5. ***Getting code onto your robot!***
   - Java and C++
     - If using gradleRIO, just type “./gradlew deploy” into the VS Code Console, and your code will be on the robot.
     - But wait! Your code doesn’t *do* anything yet. Some simple examples for drive code can be found here. The code in the snippets belong in either Robot.java, or Robot.cpp, which should be auto-created with the gradle/Eclipse project.
   - LabVIEW
     - For development, you should run from source as shown here.
     - Once complete, you will deploy a built executable as shown here.

6. ***Code for mechanisms***
   - For Java and C++, simple drive code can be copy-pasted from here.
   - For LabVIEW, the simple drive code is in the template in the TeleOp VI.

- Most *FIRST* Robotics Competition robots have actuated mechanisms other than the drivetrain. This could be anything from a spinning flywheel to a pneumatic catapult. All these mechanisms should be controllable in autonomous, or in teleop. To control mechanisms using speed controllers over PWM, there is a guide for C++ and Java here, and for LabVIEW, here.
- If using speed controllers over CAN, you must either follow the guide here to treat them as PWM speed controllers, or use the Phoenix API, whose documentation is linked here.

7. ***Autonomous***
   - A guide for how to do autonomous actions in Java and C++ programming can be found here.
   - Team 1619 has also compiled some simple code to cross the auto line in Java, which can be found here.
   - LabVIEW templates include autonomous code for jiggling the robot in place. You can modify motor power values and timing to accomplish many tasks. Here is an example of 2468's autonomous code from 2018.

Level Two: Custom Architecture, and Closed-loop Motor Control

1. ***Using a custom architecture***
   - Many times, the available robot classes are not enough. For example, you might want to run teleop periodically, and autonomous sequentially. If this is the case, it is likely time to move to a custom architecture.
   - A custom architecture is essentially structuring all the code in a customized way.
   - Some examples of custom architectures include 1678's code which is here. 1678's code builds off of 971's code which is here.
   - 254 also has a custom architecture. Their 2019 code can be found here.
   - 33, 624, and 1986, and 2468

2. ***PID Control***
   - PID Control allows you to control a mechanism based on position, rather than voltage. Using PID, you can tell an arm to turn to 30 degrees, instead of telling it to directly output a voltage. This is especially useful in autonomous. Being able to tell a robot to drive 5 metres instead of full power for 0.5 seconds allows for enhanced repeatability.
   - Some useful documents for PID are:
     - Wesley's Blog
     - CSIM's PID for Dummies

3. ***Motion Magic (CAN Only)***

- If using a TalonSRX speed controller, it is recommended to use MotionMagic for controlling mechanisms, especially something like an arm or an elevator. MotionMagic is essentially a 1KHz PID loop following autogenerated trapezoidal motion profiles. If those words make no sense, don't worry! See the above for information on PID, and here's a document explaining motion profiles.
- The documentation for Motion Magic is located here.

Level Three: Advanced Drive Paths, MP Control, and Unit-testing

1. ***Drive paths and following them***
    - Sometimes raw PID isn't enough for controlling the drivetrain autonomously. For example, you might want the robot to go around the switch and pick up a cube from behind. A clean way to do this would be to create a drive path. A drive path is essentially a set of points that the drivetrain PID loop will follow, and the points will lead to the eventual goal. PathWeaver is a graphical tool that uses a library called PathFinder which generates such paths and saves them to a parsable file. Detailed instructions on PathWeaver usage are found here.
    - Once the points have been generated, there are a variety of ways to follow them. These range from using PID to directly follow the points, to adding a path following algorithm to process the points before giving them to the PID loop. An example of such a path following algorithm can be found here (eqn 5.12). Other popular approaches to path following include adaptive pure pursuit control. 254 has a handy implementation of adaptive pure pursuit which can be found here.

2. ***Model based control***
    - Model based control is a step beyond PID. It allows for keeping a mathematical model of the system in the code, and updating the model with sensor data. Using such a model, one can control a mechanism's position, velocity, acceleration, etc much more precisely. Some teams that use model based control include 1678 and 971.
    - Useful resources for learning model-based control are:
        - Wesley's Blog
        - This MIT handout

3. ***Unit-testing***
    - Often-times, you want to test your code before deploying it on the robot. This can prevent disaster. Unit-testing is a term for testing portions of the code as standalone programs. For example, you might want to test the portion of the code that runs the elevator, but not the part that makes a few lights flash. Testing mechanisms for *FIRST* Robotics Competition is greatly enhanced with model based control, as the model can be used as a simulation of the

mechanism, meaning that the whole mechanism can be tested with incredible robustness. Some useful unit testing libraries include:

- GoogleTest

About The Compass Alliance

The Compass Alliance was founded by 10 teams from around the world with the mission of helping *FIRST* Robotics Competition teams sustain and grow. A growing Resource Repository, and 24/7 Call Center give anyone of any skill level the tools to learn something new or learn more from anywhere in the world. Remote teams lacking mentors can sign up for a Tag Team to be their remote guide throughout the season, and Help Hubs pinpoint where to gain access to local services other *FIRST* teams offer. Hear For You provides the resources and tools to help teams and volunteers develop mental wellness on their teams and at events. You can learn more about The Compass Alliance, find quality assistance, and get involved at www.thecompassalliance.org

About This Resource

This resource was prepared by The Compass Alliance, with the support and overview of *FIRST.* If you have questions about this resource, please contact thecompassalliance@gmail.com or firstroboticscompetition@firstinspires.org.

Revision History

Revision #	Revision Date	Revision Notes
1.0	Dec. 2018	Initial Release